Adaptive Flow Solver Project

AN ADAPTIVE FLOW SOLVER FOR AIR-BORNE VEHICLES UNDERGOING TIME-DEPENDENT MOTIONS/DEFORMATIONS

Annual Technical Progress Report

Period: August 1, 1996 - July 31, 1997

Prepared For

National Aeronautics and Space Administration (NASA)
Langley Research Center
Hampton, Virginia

NASA Grant No.: NAG-1-1760

NASA Technical Monitor: Roger Hathaway

By

Jatinder Singh, Ph.D. (P. I.)
Department of Engineering
Clark Atlanta University
Atlanta, Georgia

And

Stephen Taylor, Ph.D.
Department of Electrical Engineering and Computer Science
Syracuse University
Syracuse, New York

IMPLEMENTATION OF 3-D UNSTRUCTURED EULER FLOW SOLVER USING FINITE VOLUME METHODOLOGY ON HETEROGENEOUS NETWORKS

Jatinder Singh,
Department of Engineering
Clark Atlanta University
Atlanta, GA - 30314

Stephen Taylor
Department of Electrical Engineering and Computer Science
Syracuse University
Syracuse, NY

Summary

This report describes a concurrent Euler flow solver for flows around complex 3-D bodies. The solver is based on a cell-centered finite volume methodology on 3-D unstructured tetrahedral grids. In this algorithm, spatial discretization for the inviscid convective term is accomplished using an upwind scheme. A localized reconstruction is done for flow variables which is second order accurate. Evolution in time is accomplished using an explicit three-stage Runge-Kutta method which has second order temporal accuracy. This is adapted for concurrent execution using another proven methodology based on concurrent graph abstraction. This solver operates on heterogeneous network architectures. These architectures may include a broad variety of UNIX workstations and PC's running Windows NT, symmetric multiprocessors and distributed-memory multi-computers. The unstructured grid is generated using commercial grid generation tools. The grid is automatically partitioned using a concurrent algorithm based on heat diffusion. This results in memory requirements that are inversely proportional to the number of processors. The solver uses automatic granularity control and resource management techniques both to balance load and communication requirements, and deal with differing memory constraints. These ideas are again based on heat diffusion. Results are subsequently combined for visualization and analysis using commercial CFD tools. Flow simulation results are demonstrated for a constant section wing at subsonic, transonic, and a supersonic case. These results are compared with experimental data and numerical results of other researchers. Performance results are under way for a variety of network topologies.

1. Introduction

Overall thrust of this research effort is to investigate issues related to resource management, namely, implementation on heterogeneous architectures, scalability, load balancing, and numerical accuracy while solving large scale flow problems in the compressible flow domain. To achieve this end, a baseline flow solver algorithm is selected which is robust and provides accurate solutions to the flow problems involving stationary grids. Present attempt deals with implementing and validating this baseline flow solver. Next phase will involve incorporation of flow and boundary adaptive grids for flows around moving/deforming bodies.

Table of Contents

OBJECTIVES:

The objectives of the research effort funded by NASA are to develop a flow solver for large-scale, three-dimensional simulations of flow around air-borne vehicles undergoing time-dependent motions/deformations, and to investigate computational science issues such as optimization techniques to improve memory and processor utilization of parallel machines. The intent is to identify a current flow solver algorithm that works and use this to write a flow solver that solves large scale industrial problems on parallel machines such as the Cray T3D and Intel Paragon, shared-memory multi-processors, and networked workstations. The outcome will be an efficient and versatile solver that is capable of solving flows around complex configurations undergoing time-dependent motions/deformations and will be capable of directly impacting NASA missions.

METHODOLOGY:

The work is being carried out by a small multidisciplinary team consisting of (Jatinder Singh - P. I.) an aerospace engineer at Clark Atlanta University (CAU) with expertise in unsteady flows around nonrigid bodies and CFD and a computer scientist (Stephen Taylor - Subcontractor) now at Syracuse University (SU) with extensive background in issues related to scalable parallel computations and large scale computations of unsteady flows around practical configurations. This effort also supports a group of graduate and undergraduate students at CAU and SU. At CAU, an **Euler flow solver** has been developed. The Euler Equations were discretized spatially using a finite volume scheme, wherein the physical domain was subdivided into tetrahedral elemental volumes and the integral equations are applied to each volume. The algorithm is same as that of references [1,2]. The goal is to have a second order accurate flow solver. At SU, emphasis is on enhancing the existing **concurrent programming framework**. Extensive amount of work in the area of large-scale concurrent simulations [3] utilizing novel dynamic load balancing algorithms has taken place at SU. These efforts have resulted in Scalable Concurrent Programming Library (SCPlib) that is portable to heterogeneous architectures, including high-performance multi-computers, shared-memory multiprocessors, PC's running Windows NT and UNIX workstations. This library provides a framework for automatic load balancing, granularity control, interactive flow visualization and is being used in the current work.

PROGRESS:

At CAU: At the initiation of the project, SCPlib components were ported to the SGI workstation at CAU. SCPlib consists of a set of libraries that include the grid library, the structures library, and the part dealing with the concurrent graph abstraction. Initial focus was on developing understanding of these library components for parallel implementation and its use by writing

simple programs. At the same time, details about the flow solver algorithm and its implementation were being finalized. Coding of the flow solver were initiated in November '95. Simultaneously, SCPlib components were being enhanced at SU (see next paragraph). With the release of newer SCPlib in March '96, the modifications were incorporated in the Euler flow solver being developed. The coding of the Euler flow solver was completed and to help debug the flow solver, initial runs of the flow solver were made using a simple problem in which flow was entering a cube and leaving its boundaries. The grid for this problem had about 20,000 tetrahedral elements and was hand crafted. All the six outer faces would allow flow to come in and leave. Earlier this year, detailed validation of the code for flows of practical importance started by a constant section NACA0012 wing of finite span to compare results around a 2-D NACA0012 airfoil for subsonic, transonic and supersonic flow conditions. Computational grids around such a configuration were first generated using a commercially available grid generator package ICEM-CFD. However, once its license expired, move was made to switch to GridTool developed at GEOLAB in NASA Langley and VGRID grid generation package developed by ViGYAN, Inc. for NASA Langley. The P.I. took training at NASA Langley in February 1997 for generating grid using these tools and then used these to generate grids around constant section NACA 0012 wing. Next step was to make the output from VGRID compatible with I/O routines of SCPlib. Details about the implementation of the Euler flow solver are documented in a report which is included as Appendix A.

At SU: Since the beginning of the project, the team at SU has completed integration of optimized load balancing strategies into the SCPlib and has quantified the performance improvements obtained using two large scale three-dimensional applications. One of these is related to a plasma simulation and the other a three-dimensional satellite simulation. This newer version of SCPlib incorporates major changes in the graph library component and simplifies the code development process. Another success has been towards resource management by enhancing SCPLib to accommodate PC's running Windows NT. Work is continuing to further develop the SCPlib and support development effort at CAU.

Student Training: Since January '96, graduate students doing M.S. in Computer Sciences have been identified and accepted to work on the project. Simultaneously, undergraduate students have also worked with the research team at CAU. Initially, these students were exposed to parallel programming framework and SCPlib by means of reading assignments followed by one on one sessions for answering any questions. One of the undergraduate student was very good resource to the group, having been exposed to the SCPlib concepts before joining this group while he did summer internship at Caltech. At SU, Jerrell Watts, the graduate student has been a

good resource person for answering many questions regarding SCPlib to both the P.I. and the students at CAU.

Publications: As an outcome of this research, a journal paper [4] by Jerrell Watts and Stephen Taylor was been submitted to the IEEE Transactions on Parallel and Distributed Systems. A copy of the paper was enclosed with the last yearly report. Recently, an abstarct has been submitted to the AIAA Fluid Dynamic Conference to be held in June 1998. A copy of the abstract is enclosed as Appendix B.

FUTURE WORK:

Having validated the Euler flow solver, following enhancements to the solver and numerical experiments are planned. Work is currently under way to have a multi-partition version running on a single processor. This will ensure proper communication across partitions on a single processor. Next step will be to have the muti-processor version running on heterogeneous architectures. During this phase of validation, same NACA 0012 wing will be used along with an ONERA M6 wing, grid files for which have been obtained from Dr. Neal T. Frink of NASA Langley Research Center and will be used to study flow around this wing for M=0.84 and α=3.06°. These runs will establish accuracy of the flow solver and at the same time validate the inter processor and inter partition communications. This will enable us to solve large scale industrial problems. Next, viscous effects will be included to have a Navier-Stokes solver and finally, flow adaptive capability will be added. In conformity with the tasks outlined in the proposal, the final goal is to have a Navier-Stokes Flow Solver developed and validated for flow around three-dimensional bodies undergoing time-dependent motions/deformations. Practical configurations of interest will be identified in consultation with Dr. Frink from NASA Langley and flow analyzed. As such this project is significant and will have an impact on aerodynamic analysis of future air-borne vehicles that may deform during flight. Also the flow solver will have a potential to facilitate development of newer configurations that use unsteady aerodynamic forces advantageously in augmenting the performance while alleviating the undesirable effects associated with separated flows of unsteady origin.

REFERENCES:

1. Frink, N. T., "Upwind Scheme for Solving the Euler Equations on Unstructured Tetrahedral Meshes," *AIAA Journal*, Vol. 30, No. 1, pp 70-77, January 1992.

2. Frink, N. T., "Recent Progress Towards a Three-Dimensional Unstructured Navier-Stokes Solver," *AIAA Paper 94-0061*, 1994.

3. Taylor, S., J. Watts, M. Rieffel, and M. Palmer, "The Concurrent Graph: Basic Technology for Irregular Problems," *IEEE Parallel and Distributed Technology,* Vol. 4, No. 2, pp 15-25, Summer 1996.

4. Watts, J., M. Rieffel, and S. Taylor, "Practical Dynamic Load Balancing for Irregular Problems", Submitted to *IEEE Transactions on Parallel and Distributed Systems,* 1995.

APPENDIX A

The baseline flow solver uses the cell-centered finite volume methodology on unstructured tetrahedral meshes as described in reference [1]. The algorithm is robust and has been used successfully to solve many flow problems on stationary grids [1-3] as well as dynamically changing grids. Reference [4] gives results for internal viscous flows through turbomachines and Reference [5] extends analysis to 2-D dynamically changing grids. Thus, this algorithm has proven to be quite versatile. For the present, we focus on the inviscid flow problems governed by the Euler equations. Mathematical formulation is described in the next section (for details, see [1, 2]) and that is followed by description of the implementation on the heterogeneous network architectures using a proven Scalable Concurrent Programming Library (SCPlib) [6-7]. Flow simulation results are demonstrated for a constant section wing at subsonic, transonic, and a supersonic case. These results are compared with numerical results of other researchers [8,9].

2. Flow Physics - Numerical Formulation

2.1 Governing Equations

Equations governing flow of compressible inviscid nonconducting adiabatic fluid in the absence of external forces are the Euler equations which describe conservation of mass, momentum, and energy. Presented in integral form for a bounded domain Ω with boundary $\partial\Omega$ these become

$$\frac{\partial}{\partial t}\int_{\Omega} \vec{Q}\, dV + \oint_{\partial\Omega} \vec{F}(\vec{Q})\cdot\vec{n}\, ds = 0 \tag{1}$$

where

$$\vec{Q} = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ e_o \end{Bmatrix} \tag{2}$$

and

$$F = \vec{F}(\vec{Q})\cdot\vec{n} = \vec{V}\cdot\vec{n}\begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ e_o + p \end{Bmatrix} + p\begin{Bmatrix} 0 \\ n_x \\ n_y \\ n_z \\ 0 \end{Bmatrix} \tag{3}$$

Here $\vec{n}$ is the unit normal vector pointed exterior to the surface $\partial\Omega$. n_x, n_y and n_z are the Cartesian components of $\vec{n}$. The Cartesian components of velocity $\vec{V}$ are u, v, and w in the x, y, and z direction respectively. e_o is the energy per unit volume. Equation (1) has been non-dimensionalized using ρ_∞^* and a_∞^* as $\rho = \rho^*/\rho_\infty^*$, $u = u^*/a_\infty^*$, $v = v^*/a_\infty^*$, $w = w^*/a_\infty^*$, $e_o = e_o^*/(a_\infty^*)^2$, and $p = p^*/\left[\rho_\infty^*(a_\infty^*)^2\right]$. Here superscript * denotes dimensional quantities and subscript ∞ represents free stream condition. With the ideal gas assumption, pressure and total enthalpy can be expressed by

$$
\begin{aligned}
p &= (\gamma-1)\left[e_o - \frac{1}{2}\rho\left(u^2+v^2+w^2\right)\right] \\
h_o &= \frac{\gamma}{(\gamma-1)}\frac{p}{\rho} + \frac{1}{2}\left(u^2+v^2+w^2\right)
\end{aligned}
\tag{4}
$$

here γ is the ratio of specific heats and is 1.4 for air.

2.2 Spatial Discretization

A finite-volume discretization is used in the spatial domain. Equation (1) is applied to each cell. The state variables $\vec{Q}$ are volume-averaged values which are in balance with the area-averaged fluxes across the cell faces. The solution algorithm consists of essentially four steps. These are

Higher-Order Reconstruction: Given cell averaged solution in each cell at time t_n extrapolate state variable $\vec{Q}$ to second order accuracy at each face;

Boundary Conditions: Apply appropriate boundary conditions to the faces that lie on a boundary;

Flux Evaluation: Using reconstructed value of the state variable, evaluate the fluxes through the faces using an upwind scheme; and

Time Evolution: Collect flux contributions in each control volume and evolve in time using a time-stepping scheme such as an explicit Runge-Kutta scheme. Result of this process is once again cell averages at time t_{n+1}. These steps are described below.

2.2.1 Higher-Order Reconstruction

If the cell-centered state variable $\vec{Q}$ is used in the evaluation of fluxes, the scheme is only first order accurate. For a higher order scheme, estimation of the state at each cell face is achieved by interpolating the solution at each time step with a Taylor series expansion in the neighborhood of each cell center. The cell-averaged solution gradient required at the cell center is evaluated using a geometrical invariant feature of the tetrahedra [2]. The resulting formula for the flow state in terms of primitive variable $\vec{q} = \{\rho \quad u \quad v \quad w \quad p\}^T$ is given by [see Figure 1]

$$
\vec{q}_{f_{1,2,3}} = \vec{q}_c + \frac{1}{4}\left[\frac{1}{3}\left(\vec{q}_{n_1} + \vec{q}_{n_2} + \vec{q}_{n_3}\right) - \vec{q}_{n_4}\right] \tag{5}
$$

Here the subscripts n_1, n_2 and n_3 denote the vertices comprising face $f_{1,2,3}$ of cell with centroid at c and n_4 is the opposite vertex to face $f_{1,2,3}$. Formula given by Equation (5) is the analytical solution to a Taylor series expansion of $\vec{q}$ from the centroid of a tetrahedral cell to the centroids of its triangular faces. The state at the vertices is evaluated using a pseudo-Laplacian weighted averaging proceedure [1];

$$
\vec{q}_{n_i} = \left(\sum_{i=1}^{N} \omega_{c,i}\vec{q}_{c,i}\right) \Big/ \left(\sum_{i=1}^{N} \omega_{c,i}\right) \tag{6}
$$

Here N = total number of cells sharing vertex n,

$\omega_{c,i}$ = weight factor calculated at centroid c of cell i sharing vertex n, and
$\vec{q}_{c,i}$ = value of the cell-averaged primitive variable given at centroid c of cell i sharing vertex n.

Weights $\omega_{c,i}$ are evaluated as

$$\omega_{c,i} = 1 + \lambda_x\left(x_{c,i} - x_n\right) + \lambda_y\left(y_{c,i} - y_n\right) + \lambda_z\left(z_{c,i} - z_n\right) \tag{7}$$

where λ_x, λ_y, and λ_z are Lagrange multipliers which are obtained as a solution to a constrained optimization problem as described in Reference [1]. Expressions for λ_x, λ_y, and λ_z as given in Reference [1] are reproduced in Appendix A for sake of completeness.

2.2.2 Boundary Conditions

Face centered boundary conditions can be defined by either a low-order approach in which cell-averaged values are assigned to the face, or a higher-order approach that utilizes Taylor series expansion (Equation 5) to construct a more accurate estimate of the state on the boundary. Implementation of the higher-order approach requires the application of pseudo-Laplacian averaging at the boundary vertices. This requires construction of ghost cells which are image cells across the exterior boundary of an adjacent interior cell. The geometric infromation for centroid of the ghost cell is provided by [1]

$$\begin{aligned} x_{gc} &= \left(x_{c,i} - x_{n_b}\right) - 2Xn_x \\ y_{gc} &= \left(y_{c,i} - y_{n_b}\right) - 2Xn_y \\ z_{gc} &= \left(z_{c,i} - z_{n_b}\right) - 2Xn_z \end{aligned} \tag{8}$$

where

$$X = \left(x_{c,i} - x_{n_b}\right)n_x + \left(y_{c,i} - y_{n_b}\right)n_y + \left(z_{c,i} - z_{n_b}\right)n_z \tag{9}$$

is the contravariant vector component of distance and subscript n_{b} denotes a boundary vertex. These coordinates are used to generate weighting factors for the psuedo-Laplacian averaging. One also needs associated flow velocities in ghost cells and for Euler equations, these are constructed as

$$\begin{aligned} u_{gc} &= u_{c,i} - 2Un_x \\ v_{gc} &= v_{c,i} - 2Un_y \\ w_{gc} &= w_{c,i} - 2Un_z \end{aligned} \tag{10}$$

where

$$U = u_{c,i}n_x + v_{c,i}n_y + w_{c,i}n_z \tag{11}$$

Using these values and by assigning same pressure and density values at the ghost cells as that of adjacent cells from across the boundary, one can reconstruct second order accurate values at boundary surfaces. These are used to prescribe appropriate boundary values on

(a) solid boundary and plane of symmetry; and
(b) farfield inflow and outflow boundaries.

Case (a). Solid boundary and plane of symmetry: Flow tangency is implemented by subtracting the component normal to the solid face from the higher-order extrapolated values. Let $\vec{q}_{f_b}$ be the extrapolated values of the primitive variables to the boundary faces defined as

$$\vec{q}_{f_b} = \left\{\rho_{f_b} \quad u_{f_b} \quad v_{f_b} \quad w_{f_b} \quad p_{f_b}\right\}^T \tag{12}$$

and let

$$U_{f_b} = u_{f_b} n_x + v_{f_b} n_y + w_{f_b} n_z \tag{13}$$

then boundary values satisfying flow tangency conditions are computed as

$$\begin{aligned} u_b &= u_{f_b} - U_{f_b} n_x \\ v_b &= v_{f_b} - U_{f_b} n_y \\ w_b &= w_{f_b} - U_{f_b} n_z \\ p_b &= p_{f_b} \\ \rho_b &= \rho_{f_b} \end{aligned} \tag{14}$$

Then the state vector $\vec{Q}$ is calculated in terms of the conserved variables as

$$\vec{Q}_b = \left\{\rho_b \quad \rho_b u_b \quad \rho_b v_b \quad \rho_b w_b \quad e_{o_b}\right\}^T \tag{15}$$

where

$$e_{o_b} = \frac{p_b}{\gamma - 1} + \frac{\rho_b}{2}\left(u_b^2 + v_b^2 + w_b^2\right) \tag{16}$$

and flux at the boundary face is specified as

$$\vec{F}_b = p_b \left\{0 \quad n_x \quad n_y \quad n_z \quad 0\right\}_b^T \tag{17}$$

Case (b). Farfield boundary: For subsonic flows, characteristic boundary conditions are applied using the fixed and extrapolated Riemann invarients $R^{\pm}$. Since the normal to the boundary is defined as being pointed outwards, the incoming invarient R^- is determined from the freestream flow and the outgoing invarient R^+ is extrapolated from the interior domain as

$$\begin{aligned} R^- &= U_\infty - \frac{2a_\infty}{\gamma - 1} \quad ; \quad a_\infty = \left(\frac{\gamma p_\infty}{\rho_\infty}\right)^{\frac{1}{2}} \\ R^+ &= U_i - \frac{2a_i}{\gamma - 1} \quad ; \quad a_i = \left(\frac{\gamma p_i}{\rho_i}\right)^{\frac{1}{2}} \end{aligned} \tag{18}$$

Using these, locally normal velocity components and speed of sound are calculated as

$$\begin{aligned} U_{ob} &= \frac{1}{2}\left[R^+ + R^-\right] \\ a_{ob} &= \frac{\gamma - 1}{4}\left[R^+ - R^-\right] \end{aligned} \tag{19}$$

here subscript 'ob' implies value at outer boundary. Two cases are possible. If $U_{ob} > 0$, it is an outflow boundary and primitive variables at the outflow boundary are calculated by extrapolating two tangential velocities from the interior with the result

$$
\begin{aligned}
\rho_{ob} &= \left(\frac{a_{ob}}{\gamma s_{ob}}\right)^{\frac{1}{\gamma-1}} \\
u_{ob} &= u_i + n_x\left(U_{ob} - U_i\right) \\
v_{ob} &= v_i + n_y\left(U_{ob} - U_i\right) \\
w_{ob} &= w_i + n_z\left(U_{ob} - U_i\right) \\
p_{ob} &= \frac{\rho_{ob} a_{ob}^2}{\gamma}
\end{aligned}
\tag{20}
$$

where subscript i denotes interior cell values and

$$
s_{ob} = \frac{a_i^2}{\gamma\left(\rho_i\right)^{\gamma-1}} \tag{21}
$$

If $U_{ob} < 0$, it is an inflow boundary and one has

$$
\begin{aligned}
\rho_{ob} &= \left(\frac{a_{ob}}{\gamma s_{ob}}\right)^{\frac{1}{\gamma-1}} \\
u_{ob} &= u_\infty + n_x\left(U_{ob} - U_\infty\right) \\
v_{ob} &= v_\infty + n_y\left(U_{ob} - U_\infty\right) \\
w_{ob} &= w_\infty + n_z\left(U_{ob} - U_\infty\right) \\
p_{ob} &= \frac{\rho_{ob} a_{ob}^2}{\gamma}
\end{aligned}
\tag{22}
$$

where

$$
s_{ob} = \frac{a_\infty^2}{\gamma\left(\rho_\infty\right)^{\gamma-1}} \tag{23}
$$

For both of these cases, energy is calculates as

$$
e_{ob} = \frac{p_b}{\gamma - 1} + \frac{\rho_b}{2}\left(u_b^2 + v_b^2 + w_b^2\right) \tag{24}
$$

For supersonic flows, at supersonic inlet, all variables are set equal to the free stream values. At the outlet, flow variables are set equal to the interior values.

2.2.3 Flux Evaluation

To evaluate fluxes across the faces of the tetrahedral control volumes, there are two alternatives. One being the central differencing to which explicit artificial dissipation terms are added. The other one that is more popular is the upwind differencing. Upwind schemes evaluate the interface fluxes based on the characteristic theory for hyperbolic systems of equations. In these upwind schemes, information from a direction opposite to that in which the components of information are traveling is utilized. Currently, the popular upwind schemes are the flux-difference splitting (FDS) due to Roe [10], and flux-vector splitting (FVS) due to Van Leer [11]. Both schemes have been used successfully in literature. Both of these are reproduced below and will be tried and the one with least amount of computational overhead for similar quality of computational results will be used in simulating flow around deforming bodies.

2.2.3.1 Flux Vector Splitting

Flux vectors of Equation (3) are upwind differenced using the flux-vector splitting technique of van Leer [11]. Let $U = \vec{V} \cdot \vec{n} = un_x + vn_y + wn_z$ be the velocity in the direction of the outward pointing unit normal to a cell face. Also, let $M_n = U/a$ be the Mach number in the direction of U, and 'a' is speed of sound. Then, the flux vector splitting is done in terms of M_n as follows:

Case 1: $M_n \geq 1$ - Supersonic flow in the direction of a face normal,

$$F^+ = \left(\vec{F} \cdot \vec{n}\right)^+ = F \qquad ; \qquad F^- = \left(\vec{F} \cdot \vec{n}\right)^- = 0 \tag{25}$$

Here F as given by Equation (3) is evaluated using given value of U.

Case 2: $0 < M_n < 1$ - Subsonic flow in the direction of a face normal,

$$F^+ = \begin{Bmatrix} f_m^+ \\ f_m^+\{u + n_x(-U+2a)/\gamma\} \\ f_m^+\{v + n_y(-U+2a)/\gamma\} \\ f_m^+\{w + n_z(-U+2a)/\gamma\} \\ f_e^+ \end{Bmatrix} \tag{26}$$

and

$$F^- = F - F^+ \tag{27}$$

and F is given by Equation (3).

Case 3: $-1 < M_n < 0$ - Subsonic flow in the direction opposite to the face normal,

$$F^- = \begin{Bmatrix} f_m^- \\ f_m^-\{u + n_x(-U-2a)/\gamma\} \\ f_m^-\{v + n_y(-U-2a)/\gamma\} \\ f_m^-\{w + n_z(-U-2a)/\gamma\} \\ f_e^- \end{Bmatrix} \tag{28}$$

and

$$F^+ = F - F^- \tag{29}$$

Here also, F is given by Equation (3). In cases 2 and 3 above, $f_m^\pm$ and $f_e^\pm$ are given by

$$f_m^\pm = \pm \frac{\rho a}{4}(M_n \pm 1)^2 \tag{30}$$

and

$$f_e^\pm = f_m^\pm \left[\frac{(1-\gamma)U^2 \pm 2(\gamma-1)Ua + 2a^2}{(\gamma^2-1)} + \frac{u^2+v^2+w^2}{2} \right] \tag{31}$$

Case 4: $M_n \leq -1$ - Supersonic flow in the direction opposite to the face normal,

$$F^- = \left(\vec{F} \cdot \vec{n}\right)^- = F \qquad ; \qquad F^+ = \left(\vec{F} \cdot \vec{n}\right)^+ = 0 \tag{32}$$

Here also, F as given by Equation (3) is evaluated using given value of U.

Equations (25) through (29) and (32) define the split fluxes at the centroid of a cell based on reconstructed values of the state variable $\bar{Q}$. Flux through a face between two cells is evaluated as follows:

Let nodes 1, 2, and 3 define a face between two cells with centroids at 'a' and 'b'. Let the flow direction be from 'a' to 'b' and F_a^+ and F_a^- be the split fluxes along normal to face 1,2,3 for cell with centroid at 'a', and F_b^+ and F_b^- be the split fluxes along the same normal from cell 'a' to face 1,2,3 for cell 'b'. Then flux through face 1,2,3 for cell 'a' is given by [see Figure 2]

$$F_{f_{1,2,3}} = F_a^+ + F_b^- \tag{33}$$

2.2.3.2 Flux Difference Splitting

The FDS technique due to Roe [10] reconstructs the fluxes by determining an approximate solution to a Riemann problem. The fluxes across each cell face k(i) of tetrahedral cell i is computed using numerical flux formula

$$F_{i,k(i)} = \frac{1}{2}\left[F(Q_L) + F(Q_R) - |\tilde{A}|(Q_R - Q_L)\right]_{k(i)} \tag{34}$$

Here Q_L and Q_R are the conserved variables to the left and right of the interface k(i). Averaged Jacobian matrix $|\tilde{A}|$ is calculated based on following averaged quantities

$$\begin{aligned}
\tilde{\rho} &= \sqrt{\rho_L \rho_R} \\
\tilde{u} &= \left(u_L + u_R\sqrt{\rho_R/\rho_L}\right)/\left(1+\sqrt{\rho_R/\rho_L}\right) \\
\tilde{v} &= \left(v_L + v_R\sqrt{\rho_R/\rho_L}\right)/\left(1+\sqrt{\rho_R/\rho_L}\right) \\
\tilde{w} &= \left(w_L + w_R\sqrt{\rho_R/\rho_L}\right)/\left(1+\sqrt{\rho_R/\rho_L}\right) \\
\tilde{h}_o &= \left(h_{oL} + h_{oR}\sqrt{\rho_R/\rho_L}\right)/\left(1+\sqrt{\rho_R/\rho_L}\right) \\
\tilde{a}^2 &= (\gamma-1)\left[\tilde{h}_o - \left(\tilde{u}^2+\tilde{v}^2+\tilde{w}^2\right)/2\right]
\end{aligned} \tag{35}$$

Using these averaged quantities, flux $F_{i,k(i)}$ is obtained as

$$F_{i,k(i)} = \frac{1}{2}\left[F(Q_L) + F(Q_R) - |\Delta\tilde{F}_1| - |\Delta\tilde{F}_4| - |\Delta\tilde{F}_5|\right]_{k(i)} \tag{36}$$

where

$$|\Delta\tilde{F}_1| = |\tilde{U}|\left\{\left(\Delta\rho - \frac{\Delta p}{\tilde{a}^2}\right)\begin{bmatrix} 1 \\ \tilde{u} \\ \tilde{v} \\ \tilde{w} \\ \frac{\tilde{u}^2+\tilde{v}^2+\tilde{w}^2}{2} \end{bmatrix} + \tilde{\rho}\begin{bmatrix} 0 \\ \Delta u - n_x\Delta U \\ \Delta v - n_y\Delta U \\ \Delta w - n_z\Delta U \\ \tilde{u}\Delta u + \tilde{v}\Delta v + \tilde{w}\Delta w - \tilde{U}\Delta U \end{bmatrix}\right\} \tag{37}$$

and

$$\left|\Delta\tilde{F}_{4,5}\right| = \left|\tilde{U} \pm \tilde{a}\right|\left(\frac{\Delta p \pm \tilde{\rho}\tilde{a}\Delta U}{2\tilde{a}^2}\right)\begin{bmatrix} 1 \\ \tilde{u} \pm n_x\tilde{a} \\ \tilde{v} \pm n_y\tilde{a} \\ \tilde{w} \pm n_z\tilde{a} \\ \tilde{h}_o \pm \tilde{U}\tilde{a} \end{bmatrix} \tag{38}$$

where $\tilde{U} = \tilde{u}n_x + \tilde{v}n_y + \tilde{w}n_z$, $\Delta U = n_x\Delta u + n_y\Delta v + n_z\Delta w$ and the operator Δ in the right hand side of Equations (37) and (38) is defined as $\Delta(\ \ ) = (\ \ )_R - (\ \ )_L$.

2.2.4 Time Evolution

After discretizing Equation (1) in space, the following system of coupled ordinary differential equations (ODE) is obtained:

$$V_i \frac{dQ_i}{dt} + R_i = 0 \qquad , \qquad i = 1,2,3\cdots \tag{39}$$

where

$$R_i = \sum_{j=1}^{k(i)} F_{i,j}\,\Delta S_{i,j} \tag{40}$$

is summation of the fluxes through the four faces k of the tetrahedral cell i. $F_{i,j}$ is the flux value through face j of cell i provided by Equations (33) or (34), $\Delta S_{i,j}$ is the area of face j of cell i, and V_i is the volume of cell i.

As stated before, main objective of this research effort is to develop a flow solver that gives time-accurate solutions to unsteady flow problems. In order to solve the system of ODE's defined by Equation (38), one could use either explicit or implicit schemes. The choice depends on the time scales of the physical unsteady phenomena under investigation and to compare it with the time step restriction arising from the numerical scheme, e.g., CFL condition number for explicit scheme.

Let Δt_p be the time-step limited by physics, Δt_n be the time-step limited by numerics (e.g., CFL condition number for explicit scheme), and t_r be the ratio of these time steps $\left(= \Delta t_p/\Delta t_n\right)$.

If $t_r >> 1$, then large number of Δt_n would be required to cover a physical time step Δt_p. In this case, implicit methods should be used. However, if $t_r \leq 1$; then explicit schemes are a way to go since Δt_p is smaller than Δt_n and there is no advantage in using implicit scheme which entail relatively large storage requirements as compared to low storage explicit schemes such as m-stage Runge-Kutta method [3].

As a first step, an explicit scheme namely multi-stage Runge-Kutta method will be used. Using a m-stage scheme provides m^{th} order accuracy for linear systems. Both 3-stage and 4-stage schemes are only 2^{nd} order accurate for the given system of nonlinear equations [9]. 3-stage scheme has been used in References [1-3]. A 4-stage scheme allows for a CFL

condition number close to 3. This will allow larger time step as compared to a 3-stage scheme at the cost of one additional set of updates per time step. The m-stage Runge-Kutta time-stepping scheme can be written as

$$\begin{aligned} Q_i^{(0)} &= Q_i^n \\ Q_i^{(1)} &= Q_i^{(0)} - \alpha_1 \frac{\Delta t}{V_i} R_i^{(0)} \\ &\cdots \\ Q_i^{(m-1)} &= Q_i^{(0)} - \alpha_{m-1} \frac{\Delta t}{V_i} R_i^{(m-2)} \\ Q_i^{(m)} &= Q_i^{(0)} - \alpha_m \frac{\Delta t}{V_i} R_i^{(m-1)} \\ Q_i^{n+1} &= Q_i^{(m)} \end{aligned} \tag{41}$$

where for a 3-stage scheme,

$$\alpha_1 = \frac{1}{3} \qquad \alpha_2 = \frac{1}{2} \qquad \alpha_3 = 1$$

As a first trial, 3-stage scheme will be used.

If the goal of the computational experiment is to reach steady state solution to a problem, it is possible to use some acceleration techniques such as local time-stepping and/or implicit residual smoothing at each time to accelerate convergence to steady state. The results obtained after each time-step when using such accelerating schemes are not time-accurate from unsteady flow point of view. Thus, if the goal is to have time-accurate solutions for unsteady flow problems, these acceleration techniques should not be used. However, initially for validation of the flow solver, we will use local time stepping and implicit residual smoothing. These are described below.

2.2.4.1 Local Time Stepping

Local time stepping accelerates convergence by advancing the solution at each cell in time at a CFL number near the local stability limit. The expression for the local time step was derived with the aid of a 2-D stability analysis.

$$\Delta t \le \nu \frac{V_i}{A_i + B_i + C_i} \tag{42}$$

with

$$\begin{aligned} A_i &= \left(|u_i| + a_i\right) S_i^{(x)} \\ B_i &= \left(|v_i| + a_i\right) S_i^{(y)} \\ C_i &= \left(|w_i| + a_i\right) S_i^{(z)} \end{aligned} \tag{43}$$

where ν is the CFL number, V_i is the cell volume, a_i is the local speed of sound, and $S_i^{(x)}$, $S_i^{(y)}$, and $S_i^{(z)}$ are projected areas of cell i in the x, y, and z directions.

2.2.4.2 Implicit Residual Smoothing

The maximum time step can be further increased by increasing the support of the scheme through implicit averaging of the residual with their neighbors. The residuals are filtered through a smoothing operator (which is essentially the Laplacian operator for a uniform grid):

$$\overline{R}_i = R_i + \varepsilon \nabla^2 \overline{R}_i \tag{44}$$

where

$$\nabla^2 \overline{R}_i = \sum_{j=k(i)} \left(\overline{R}_j - \overline{R}_i \right) \tag{45}$$

The summation uses difference in the residual from neighboring cells that share the four faces of the tetrahedra. The resulting equations are solved using Jacobi iterations and for the present case of tetrahedral grids and with recommended value of $\varepsilon = 0.5$, the equation becomes

$$\overline{R}_i^{(m)} = \frac{1}{3}\left(R_i + 0.5 \sum_{j=1}^{4} \overline{R}_j^{(m-1)} \right) \tag{46}$$

Two Jacobi iterations have been found satisfactory and the residual smoothing is repeated during every stage of the Runge-Kutta time cycle.

3. Concurrent Implementation of the Flow Solver

The concurrent algorithm is based on a domain decomposition that divides the grid into partitions. Each partition is solved independently using appropriate boundary conditions such as presence of a body, inflow/outflow and so on. Boundaries at the partition cut represents a nonphysical boundary and information from adjacent boundaries is communicated between partitions to solve flow in those areas. This algorithm is implemented using the Scalable Concurrent Processing Library (SCPlib) [6]. This library supports irregular applications on scalable concurrent hardware over heterogeneous networks. With this library, an application is implemented as a graph comprising nodes and directed edges. The nodes correspond to partitions of the problem, and edges correspond to communication channels (Figure 3). Multiple nodes are mapped to a single processor, or to a collection of processors sharing memory.

Each node has four components (Figure 4). A node's *state* is the set of variables or data structures that represent a problem partition. In the present problem, *state* is described by flow variables and flux through boundaries and associated data structures. A collection of application specific *physics* routines are implemented in each partition. These correspond to implementation of the numerical formulation for the Euler equations. The *communication list* describes the mapping of nodes to processors and is used to send messages between nodes. These are built during the partitioning phase and represent data dependencies in the numerical scheme. These dependencies describe values to be extracted from the state sent between nodes at each iteration. Finally, there are *other* functions which are application and architecture independent but that function under the assumption that the computations conforms to the graph's architecture. The library provides these functions and they accomplish important tasks such as load balancing, granularity control and visualization. This library has been used to implement the Euler flow solver.

Figure below shows the abstract algorithm for the Euler flow solver, in terms of the Scalable Concurrent Processing Library (SCPlib).

```
partition(.....)
{   load geometry data into partition
    initialize state
    calculate local Δt and norm
    gather/scatter to obtain global norm
    while(termination criteria not met) {
        extract state at partition boundaries
        send state at partition boundaries to neighbors
        receive state from neighboring boundaries
        compute state at newer iteration
        calculate new local Δt and norm
        gather/scatter to obtain new global norm
    }
}
```

Concurrent Euler flow solver algorithm

Node's physics routines are encapsulated behind the interfaces provided by the initialize, extract and compute functions. The last function receives the data during communications and subsequently solves the Euler equations for a single partition at a given time-step/iteration. This function is essentially the sequential version of the Euler flow solver with the boundary condition that represents a cut in the domain.

4. Flow Results

In order to validate the flow solver, several test cases were run on a single processor of a computer/ workstation. A simple geometry was selected consisting of a constant section NACA 0012 wing with a unit chord and 0.1 semi-span. The motivation was to solve 3-D flow and compare results with standard 2-D cases at subsonic(sub-critical), transonic, and supersonic flow regimes. Grid was generated using GridTool/VGRID grid generation software developed at NASA Langley Research Center. Grid consisted of 5996 vertices, 9588 faces and 20815 tetrahedral elements. It should be noted that the grid used in this study is coarse as compared to the two-dimensional grid used in reference [8]. The computational domain is bounded by a rectangular box with boundaries at $-8 \le x \le 12$, $0 \le y \le 0.1$ and $-8 \le z \le 8$. Figures 5 and 6 show respectively, the near-field and the far-field grid at the symmetry plane.

Computations were carried out for the test cases of a) subsonic (subcritical) M=0.63, $\alpha = 2°$, b) transonic M=0.85, $\alpha = 1°$, and c) supersonic M=1.2, $\alpha = 0°$. Results are presented for each of these case by taking a cut plane at the mid wing location (y=0.05). Results obtained from the cut plane are compared with the 2-D computations of reference [8]. Computations were carried out on SGI Octane and it took about 10 hours of cpu time to carry out 5000 iterations for each case. For the subsonic (subcritical) case, Iso-Mach lines are presented in

Figure 7 with ($\Delta M = 0.028$) and compared with the computations from [8] and show good comparison. Figures 8 and 9 shows respectively Cp distribution and Mach number on the surface of the airfoil and Figure 10 shows Mach number distribution in the cut plane. This grid resolves the subsonic (subcritical) case fairly well.

For the transonic case, Figure 11 shows the Iso-mach contours ($\Delta M = 0.05$). Both the upper and the lower surface shocks are present but are diffused because of relatively coarse grid which is unable to resolve the shock discontinuity sharply in the flowfield. Figure 12 shows the Cp distribution. It is seen that both the shocks are captured. Figure 13 shows the Mach number distribution along the wing surface at the cut plane. These values are calculated by picking face centered values of surface data where cut plane intersects the surface triangles and are assumed constant over the length of the line segment over the face belonging to the cut plane. This lower order interpolation scheme causes kinks to appear in these curves. Figure 14 shows the Mach number distribution at the cut plane.

For the supersonic case, Figure 15 shows the Iso-mach contours ($\Delta M = 0.05$). Both the bow and the trailing edge shocks are present but are diffused because of relatively coarse grid which is unable to resolve the shock discontinuity sharply in the flowfield. Figure 16 shows the Cp distribution. It compares well with the distribution of Reference [8]. Figure 17 shows the Mach number distribution along the wing surface at the cut plane. As explained above, some kinks show up in these curves because of the lower order interpolation scheme. Figure 18 shows the Mach number distribution at the cut plane.

5. Acknowledgments

The first author would like to express his indebtedness to Dr. Neal T. Frink of NASA Langley Research Center for providing technical support and Dr. Shahyar Pirzadeh of ViGYAN, Inc. and Mr. Javier A. Garriz for providing help with grid generation. This work was performed under NASA Grant NAG-1-1760 and support and encouragement from Mr. Roger Hathaway, the Technical Monitor is also acknowledged.

6. References

1. Frink, N. T., "Recent Progress Towards a Three-Dimensional Unstructured Navier-Stokes Flow Solver," AIAA Paper 94-0061, 1994.

2. Frink, N. T., P. Parikh, and S. Pirzadeh, "A Fast Upwind Solver for the Euler Equations on Three-Dimensional Unstructured Meshes," AIAA Paper 91-0102, 1991.

3. Frink, N. T., "Upwind Scheme for Solving the Euler Equations on Unstructured Tetrahedral Meshes," AIAA Journal, Vol 30, No. 1, January 1992, pp 70-77.

4. Kwon, O. J., and C. Hah, "Simulation of Three-Dimensional Turbulent Flows on Unstructured Meshes," AIAA Journal, Vol 33, No. 6, June 1995, pp 1081-1089.

5. Singh, K. P., J. C. Newman, and O. Baysal, "Dynamic Unstructured Method for Flows past Multiple Objects in Relative Motion," AIAA Journal, Vol 33, No. 4, April 1995, pp 641-649.

6. Taylor, Stephen, Jerrell R. Watts, Marc A. Rieffel, and Michael Palmer, "The Concurrent Graph: Basic Technology for Irregular Problems," IEEE Parallel & Distributed Technology, Systems & Applications, pp. 15-25, Summer 1996.

7. Watts, Jerrell, and Stephen Taylor, "A Practical Approach to Dynamic Load Balancing," Technical Report, Scalable Concurrent Programming Laboratory, Syracuse University.

8. Viviand, H., "Numerical Solutions of 2D Reference Test cases," AGARD Technical Report AR-211, May 1985.

9. Delanaye, Michel, "Polynomial Reconstruction Finite Volume Schemes for the compressible Euler and Navier-Stokes Equations on Unstructured Adaptive Grids," Ph.D. Thesis, September 1996,University of Liege.

10. Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," Journal of Computational Physics, Vol 43, 1981.

11. Van Leer, B., "Flux vector Splitting for the Euler Equations," Lecture Notes in Physics, Vol 170, 1982, pp 501.

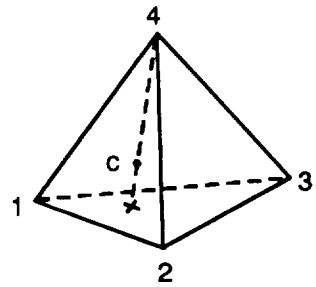

Figure 1:Notation for extrapolation based on higher order averaging in a tetrahedra.

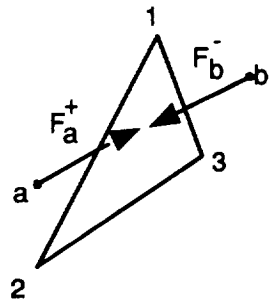

Figure 2: Flux through a face shared by two tetrahedra.

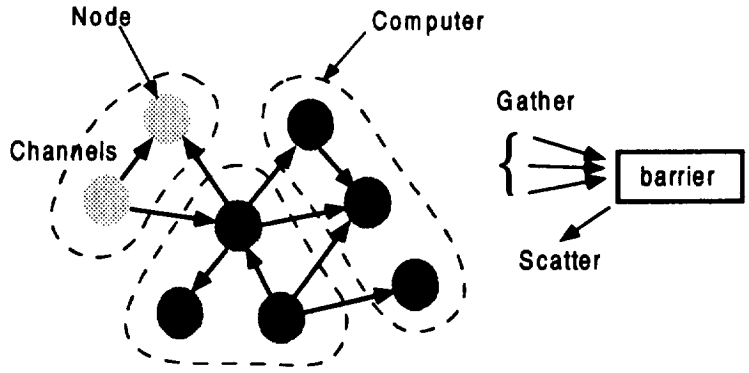

Figure 3: Concurrent graph

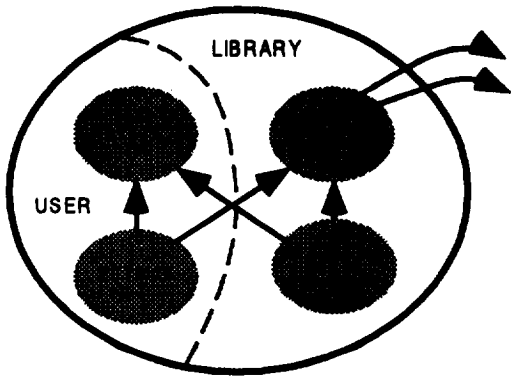

Figure 4: Graph node

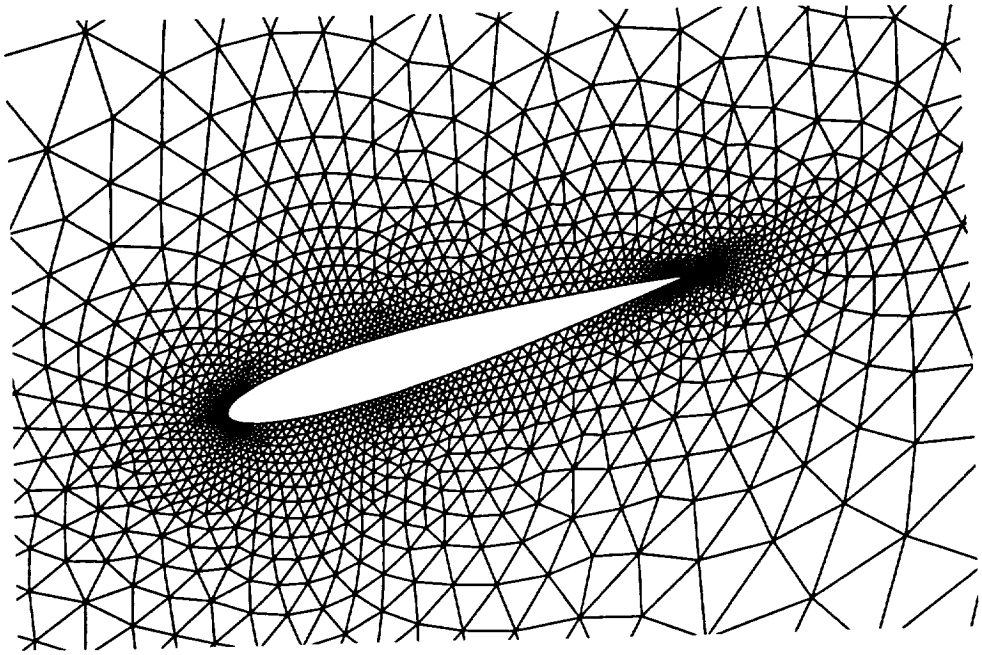

Figure 5: Near-field grid

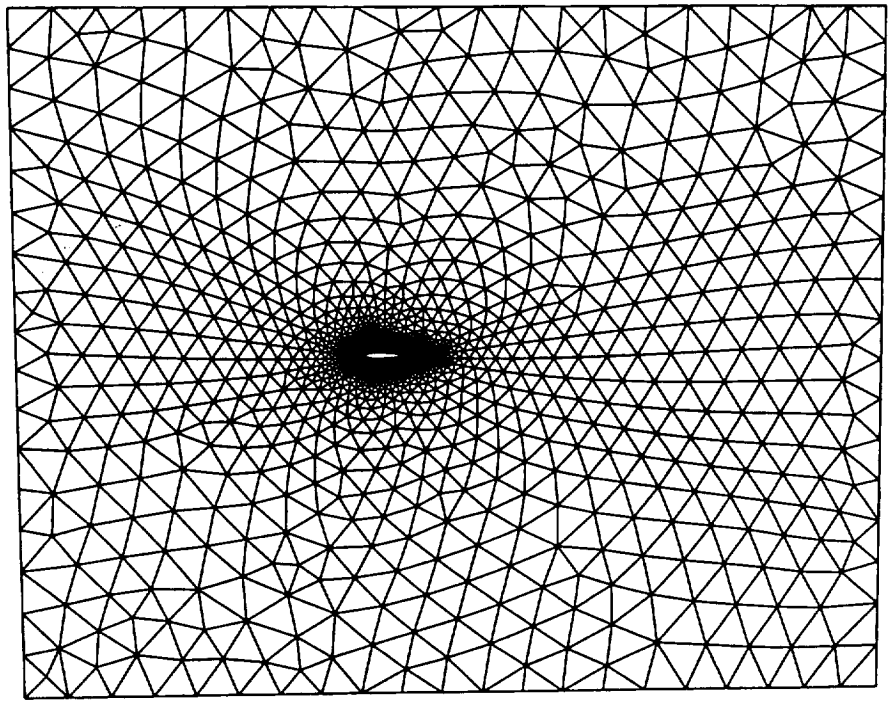

Figure 6: Far-field grid

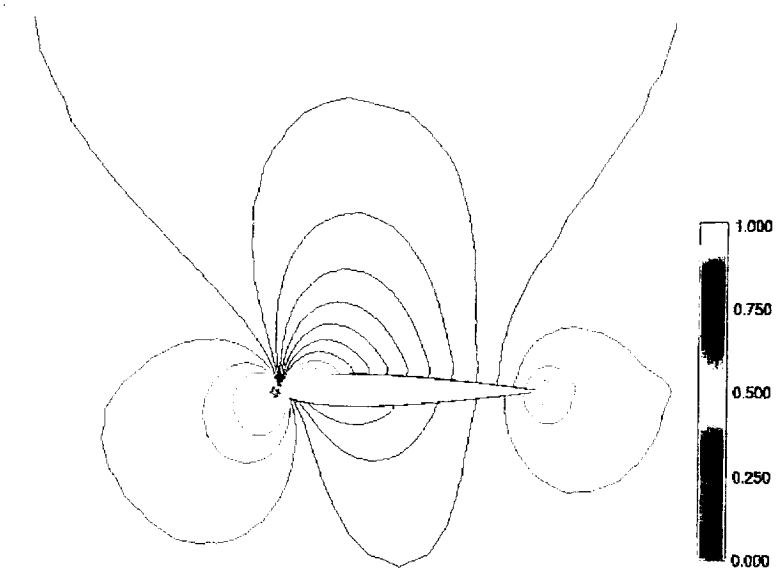

Figure 7: Iso-Mach contours for the subsonic case.

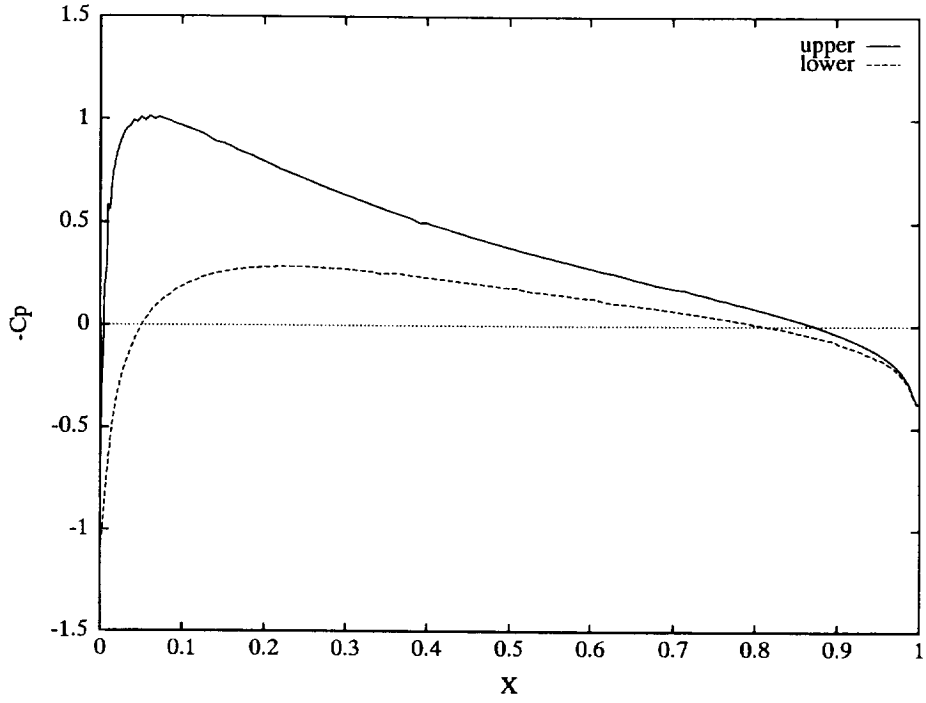

Figure 8: Cp distribution for the subsonic case.

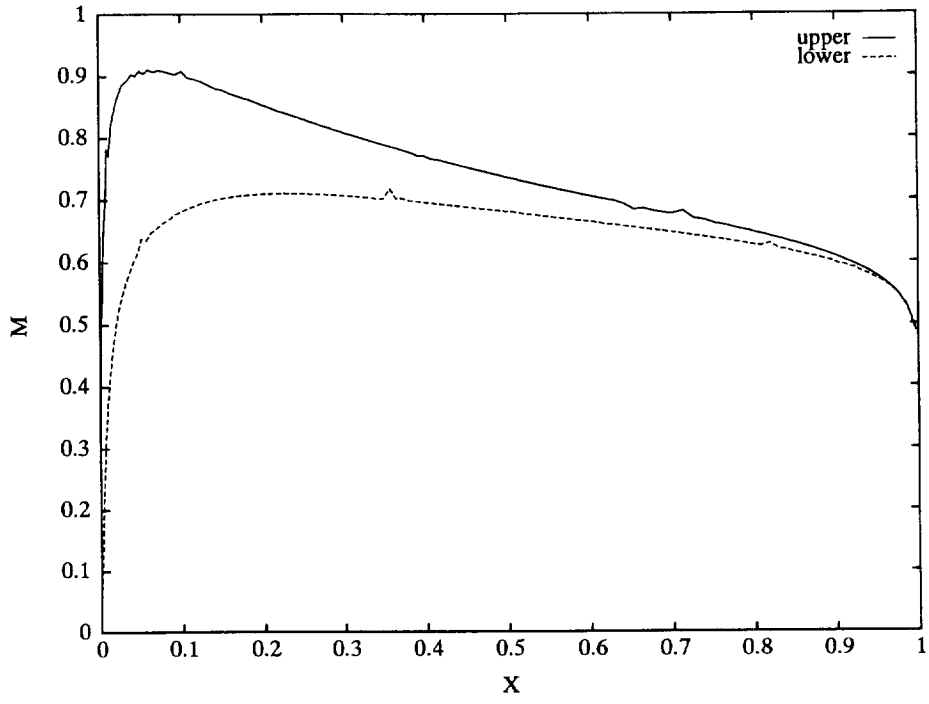

Figure 9: Mach number distribution on the surface for the subsonic case.

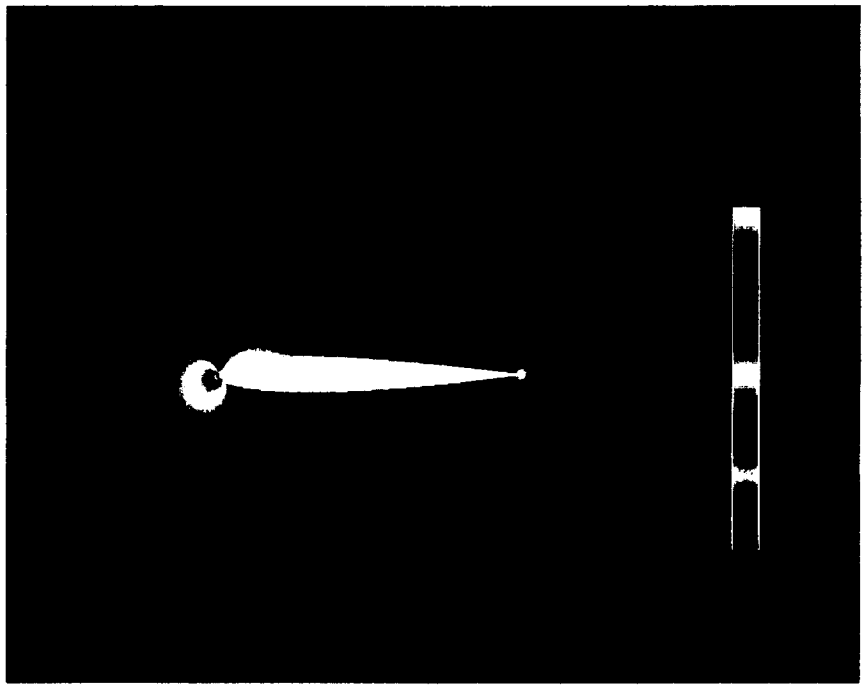

Figure 10: Mach number distribution in the cut plane for the subsonic case.

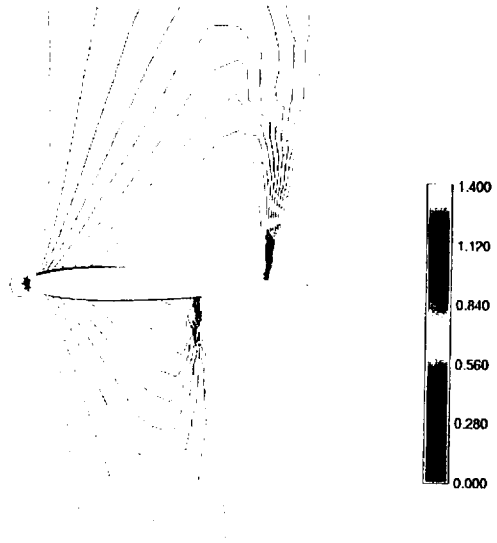

Figure 11: Iso-Mach contours on surface for the transonic case.

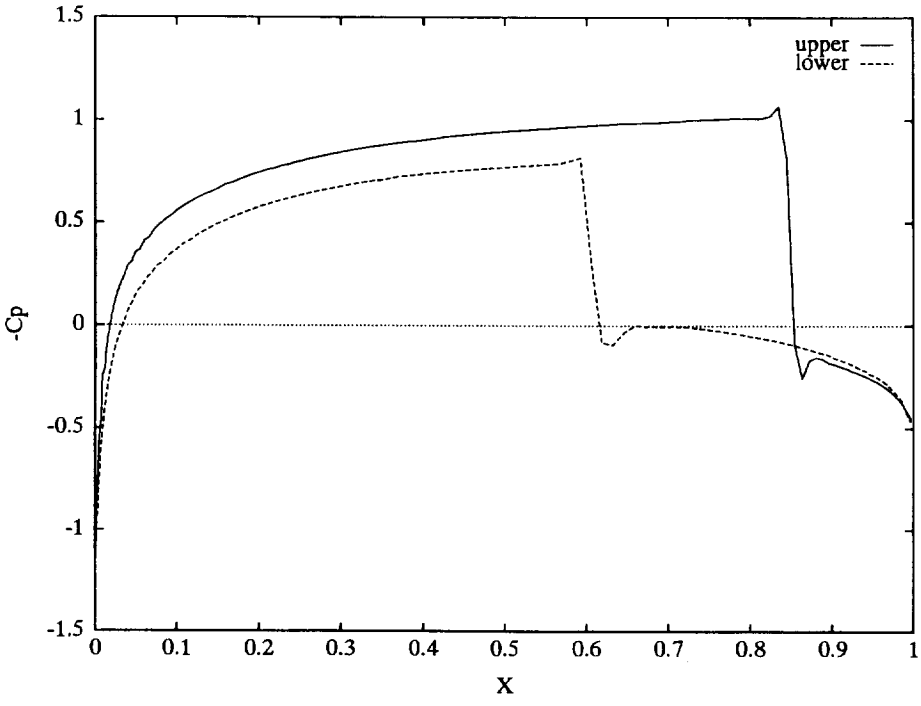

Figure 12: Cp distribution on surface for the transonic case.

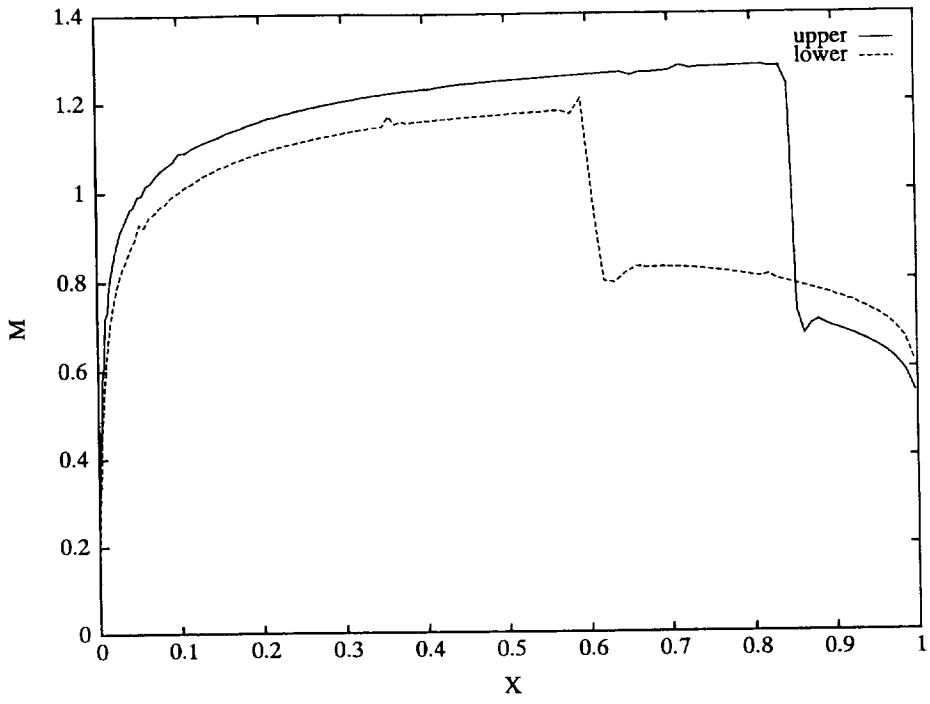

Figure 13: Mach number distribution on surface for the transonic case.

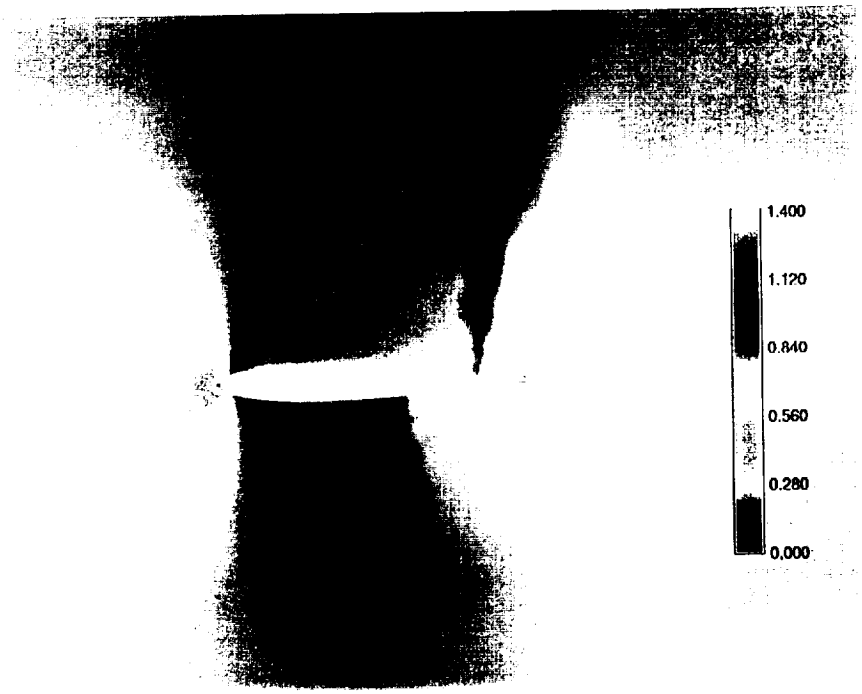

Figure 14: Mach number distribution in the cut plane for the transonic case.

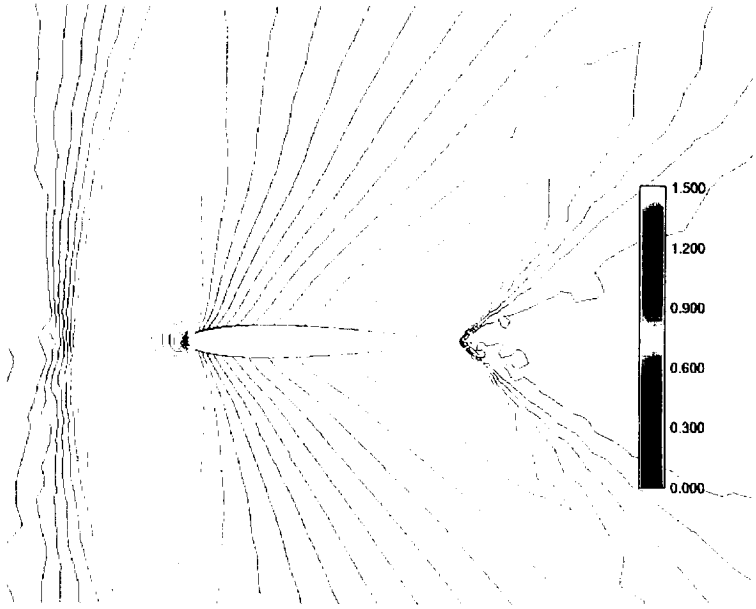

Figure 15: Iso-Mach contours on surface for the supersonic case.

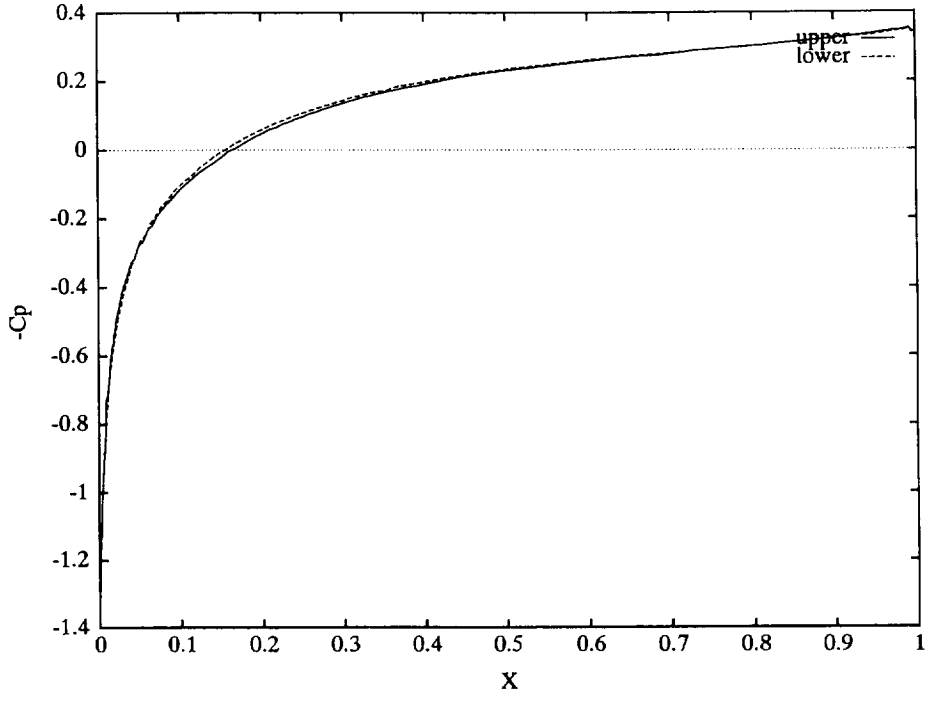

Figure 16: Cp distribution on surface for the supersonic case.

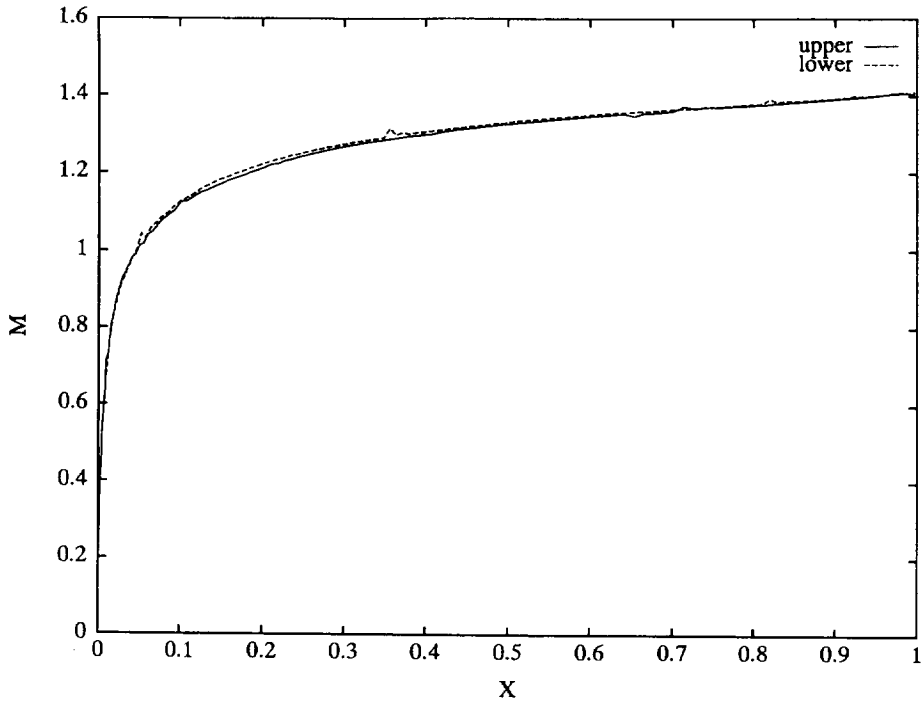

Figure 17: Mach number distribution on surface for the supersonic case.

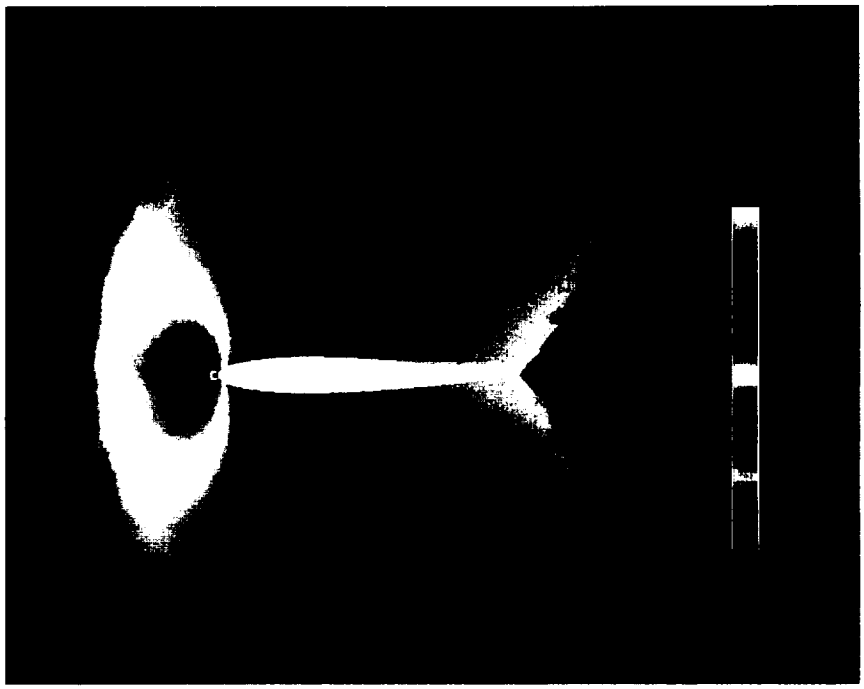

Figure 18: Mach number distribution in the cut plane for the supersonic case.

Appendix A

Expressions for Lagrange Multipliers λ_x, λ_y, and λ_z

Given coordinates of a node n, and coordinates of N centroids of all tetrahedra sharing node n, Lagrange multipliers λ_x, λ_y, and λ_z are calculated as follows. First some intermediate values are defined as

$$R_x = \sum_{i=1}^{N} \left(x_{c,i} - x_n\right)$$

$$R_y = \sum_{i=1}^{N} \left(y_{c,i} - y_n\right)$$

$$R_z = \sum_{i=1}^{N} \left(z_{c,i} - z_n\right)$$

$$I_{xx} = \sum_{i=1}^{N} \left(x_{c,i} - x_n\right)^2$$

$$I_{yy} = \sum_{i=1}^{N} \left(y_{c,i} - y_n\right)^2$$

$$I_{zz} = \sum_{i=1}^{N} \left(z_{c,i} - z_n\right)^2$$

$$I_{xy} = \sum_{i=1}^{N} \left(x_{c,i} - x_n\right)\left(y_{c,i} - y_n\right)$$

$$I_{xz} = \sum_{i=1}^{N} \left(x_{c,i} - x_n\right)\left(z_{c,i} - z_n\right)$$

$$I_{yz} = \sum_{i=1}^{N} \left(y_{c,i} - y_n\right)\left(z_{c,i} - z_n\right)$$

$$D = I_{xx}\left(I_{yy}I_{zz} - I_{yz}^2\right) - I_{xy}\left(I_{xy}I_{zz} - I_{xz}I_{yz}\right) + I_{xz}\left(I_{xy}I_{yz} - I_{yy}I_{xz}\right)$$

and the Lagrange multipliers are calculated as

$$\lambda_x = \left[-R_x\left(I_{yy}I_{zz} - I_{yz}^2\right) + R_y\left(I_{xy}I_{zz} - I_{xz}I_{yz}\right) - R_z\left(I_{xy}I_{yz} - I_{yy}I_{xz}\right)\right] / D$$

$$\lambda_y = \left[R_x\left(I_{xy}I_{zz} - I_{xz}I_{yz}\right) - R_y\left(I_{xx}I_{zz} - I_{xz}^2\right) + R_z\left(I_{xx}I_{yz} - I_{xy}I_{xz}\right)\right] / D$$

$$\lambda_z = \left[-R_x\left(I_{xy}I_{yz} - I_{yy}I_{xz}\right) + R_y\left(I_{xx}I_{yz} - I_{xy}I_{xz}\right) - R_z\left(I_{xx}I_{yy} - I_{xy}^2\right)\right] / D$$

APPENDIX B

An Unstructured Euler Solver for Heterogeneous Networks

Jatinder Singh[1], John Maweu[2], Jerrell Watts[3], and Stephen Taylor[4]

Summary

This paper describes a concurrent Euler flow solver for flows around complex 3-D bodies. The solver is based on a proven, finite-volume methodology that has second-order accuracy and is adapted for concurrent execution using another proven methodology based on concurrent graph abstraction. This solver operates on heterogeneous network architectures. These architectures may include a broad variety of UNIX workstations and PC's running Windows NT, symmetric multiprocessors and distributed-memory multi-computers. The grid is generated using commercial grid generation tools. The grid is automatically partitioned using a concurrent algorithm based on heat diffusion. This results in memory requirements that are inversely proportional to the number of processors. The solver uses automatic granularity control and resource management techniques both to balance load and communication requirements, and deal with differing memory constraints. These ideas are again based on heat diffusion. Results are subsequently combined for visualization and analysis using commercial CFD tools. Flow simulation results are demonstrated for a constant section wing at subsonic, transonic, and a supersonic case. These results are compared with experimental data and numerical results of other researchers. Performance results are under way for a variety of network topologies.

1. Introduction

Overall thrust of this research effort is to investigate issues related to resource management, namely, scalability, load balancing, and numerical accuracy while solving large scale flow problems in the compressible flow domain. To achieve this end, a baseline flow solver algorithm is selected which is robust and provides accurate solutions to the flow problems involving stationary grids. Present attempt deals with implementing and validating this baseline flow solver. The baseline flow solver uses the cell-centered finite volume methodology on unstructured tetrahedral meshes as described in reference [1]. The algorithm is robust and has been used successfully to solve many flow problems on stationary grids [1-3] as well as dynamically

[1] Associate Professor, Clark Atlanta University
[2] Student Assistant, Clark Atlanta University
[3] Graduate Student Assistant, Syracuse University
[4] Associate Professor, Syracuse University

changing grids. Reference [4] gives results for internal viscous flows through turbomachines and Reference [5] extends analysis to 2-D dynamically changing grids. Thus, this algorithm has proven to be quite versatile. For the present, we focus on the inviscid flow problems governed by the Euler equations. Mathematical formulation is described briefly in the next section (for details, see [1]) and that is followed by description of the implementation on the on heterogeneous network architectures using a proven Scalable Concurrent Programming Library (SCPlib) [6,7]. Finally, results are presented and compared to work of others [8,9].

2. Flow Physics - Numerical Formulation

2.1 Governing Equations

Equations governing flow of compressible inviscid nonconducting adiabatic fluid in the absence of external forces are the Euler equations which describe conservation of mass, momentum, and energy. Presented in integral form for a bounded domain Ω with boundary $\partial\Omega$ these become

$$\frac{\partial}{\partial t}\int_{\Omega}\vec{Q}\,dV+\oint_{\partial\Omega}\vec{F}(\vec{Q})\cdot\vec{n}\,ds=0 \tag{1}$$

where $\vec{Q}=\{\rho \quad \rho u \quad \rho v \quad \rho w \quad e_o\}^T$ and

$$F=\vec{F}(\vec{Q})\cdot\vec{n}=\vec{V}\cdot\vec{n}\{\rho \quad \rho u \quad \rho v \quad \rho w \quad e_o+p\}^T+p\{0 \quad n_x \quad n_y \quad n_z \quad 0\}^T \tag{2}$$

Here $\vec{n}$ is the unit normal vector pointed exterior to the surface $\partial\Omega$. n_x, n_y and n_z are the Cartesian components of $\vec{n}$. The Cartesian components of velocity $\vec{V}$ are u, v, and w in the x, y, and z direction respectively. e_o is the energy per unit volume. Equation (1) has been non-dimentionalized using ρ_∞^* and a_∞^* as $\rho=\rho^*/\rho_\infty^*$, $u=u^*/a_\infty^*$, $v=v^*/a_\infty^*$, $w=w^*/a_\infty^*$, $e_o=e_o^*/(a_\infty^*)^2$, and $p=p^*/\left[\rho_\infty^*(a_\infty^*)^2\right]$. Here superscript * denotes dimensional quantities and subscript ∞ represents free stream condition. With the ideal gas assumption, pressure and total enthalpy can be expressed by

$$\begin{aligned} p &= (\gamma-1)\left[e_o-\frac{1}{2}\rho\left(u^2+v^2+w^2\right)\right] \\ h_o &= \frac{\gamma}{(\gamma-1)}\frac{p}{\rho}+\frac{1}{2}\left(u^2+v^2+w^2\right) \end{aligned} \tag{3}$$

here γ is the ratio of specific heats and is 1.4 for air.

2.2 Spatial Discretization

A finite-volume discretization is used in the spatial domain. Equation (1) is applied to each cell. The state variables $\bar{Q}$ are volume-averaged values which are in balance with the area-averaged fluxes across the cell faces. The solution algorithm consists of essentially four steps. These are

Higher-Order Reconstruction: *Given cell averaged solution in each cell at time* t_n, *extrapolate state variable* $\vec{q} = \{\rho \; u \; v \; w \; p\}^T$ *to second order accuracy at each face.* This is accomplished by first evaluating the state at the vertices using the pseudo-Laplacian weighted averaging [1]. That is followed by doing a Taylor series expansion in the neighborhood of the cell center and extrapolating values at the face of the tetrahedra using a geometrical invariant feature of the tetrahedra [2]. This yields second order accuracy of the state variable which is used in evaluating of the flux.

Boundary Conditions: *Apply appropriate boundary conditions to the faces that lie on a boundary.* At the boundary faces, higher order reconstructed values are corrected to impose appropriate boundary conditions. At the body and the symmetry plane, flow tangency is implemented by subtracting the component normal to the face from the higher-order extrapolated values. At the farfield boundaries, characteristic boundary conditions are applied using the fixed and extrapolated Riemann invariants. Since the normal to the boundary is defined as being pointing outwards, the incoming invariant is determined from the freestream flow and the outward invariant is extrapolated from the interior domain.

Flux Evaluation: *Using reconstructed value of the state variable, evaluate the fluxes through the faces using an upwind scheme.* This is accomplished using the popular flux-difference splitting scheme due to Roe [10]. The flux across each face cell is computed from Roe-averaged quantities details of which are given in [2].

Time Evolution: *Collect flux contributions in each control volume and evolve in time.* A time-stepping scheme such as an explicit three-stage Runge-Kutta scheme is used. At each stage of the scheme, local time-stepping and implicit residual smoothing [2] are used to accelerate convergence to steady state. At the end, result of this process is once again cell averages at time t_{n+1}.

3. Concurrent Implementation of the Flow Solver

The concurrent algorithm is based on a domain decomposition that divides the grid into partitions. Each partition is solved independently using appropriate boundary conditions such as presence of a body, inflow/outflow and so on. Boundaries at the partition cut represents a nonphysical boundary and information from adjacent boundaries is communicated between partitions to solve flow in those areas. This algorithm is implemented using the Scalable Concurrent Processing Library (SCPlib) [6]. This library supports irregular applications on scalable concurrent hardware over heterogeneous networks. With this library, an application is implemented as a graph comprising nodes and directed edges. The nodes correspond to partitions of the problem, and edges correspond to communication channels (Figure 1). Multiple nodes are mapped to a single processor, or to a collection of processors sharing memory.

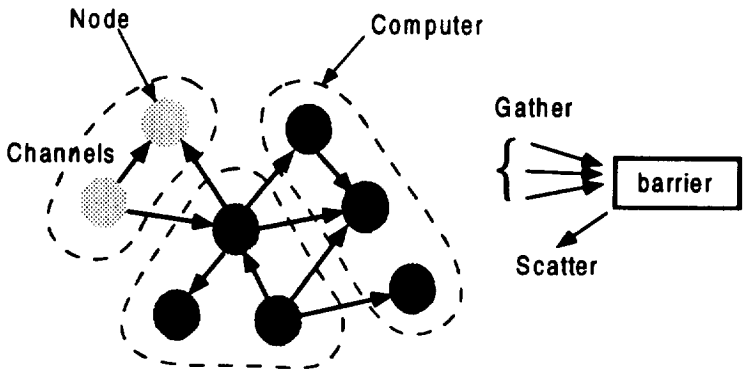

Figure 1:Concurrent graph

Each node has four components (Figure 2). A node's *state* is the set of variables or data structures that represent a problem partition. In the present problem, *state* is described by flow variables and flux through boundaries and associated data structures. A collection of application specific *physics* routines are implemented in each partition. These correspond to implementation of the numerical formulation for the Euler equations. The *communication list* describes the mapping of nodes to processors and is used to send messages between nodes. These are built during the partitioning phase and represent data dependencies in the numerical scheme. These dependencies describe values to be extracted from the state sent between nodes at each iteration. Finally, there are *other* functions which are application and architecture independent but that function under the assumption that the computations conforms to the graph's architecture. The

library provides these functions and they accomplish important tasks such as load balancing, granularity control and visualization. This library has been used to implement the Euler flow solver.

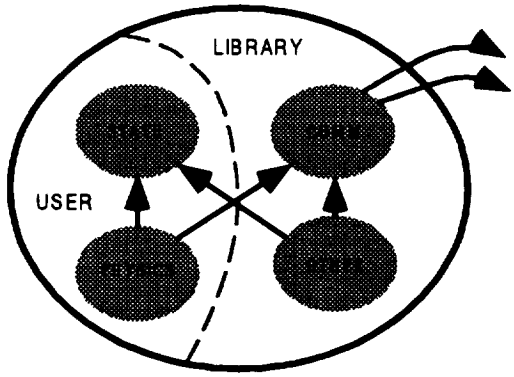

Figure 2: Graph node

Figure 3 below shows the abstract algorithm for the Euler flow solver, in terms of the Scalable Concurrent Processing Library (SCPlib).

```
partition(.....)
{   load geometry data into partition
    initialize state
    calculate local Δt and norm
    gather/scatter to obtain global norm
    while(termination criteria not met) {
        extract state at partition boundaries
        send state at partition boundaries to neighbors
        receive state from neighboring boundaries
        compute state at newer iteration
        calculate new local Δt and norm
        gather/scatter to obtain new global norm
    }
}
```

Figure 3: Concurrent Euler flow solver algorithm

Node's physics routines are encapsulated behind the interfaces provided by the initialize, extract and compute functions. The last function receives the data during communications and subsequently solves the Euler equations for a single partition at a given timestep/iteration. This function is essentially the sequential version of the Euler flow solver with the boundary condition that represents a cut in the domain.

4. Flow Results

4. Flow Results

In order to validate the flow solver, test cases were run on a single processor of a machine. A simple geometry was selected consisting of a constant section NACA 0012 wing with a unit chord and 0.1 semi-span. The motivation was to solve 3-D flow and compare results with standard 2-D cases at subsonic(subcritical), transonic, and supersonic flow regimes. Grid was generated using GridTool/VGRID grid generation software developed at NASA Langley Research center. Grid consisted of 5996 vertices, 9588 faces and 20815 tetrahedral elements. The computational domain is bounded by a rectangular box with boundaries at $-8 \le x \le 12$, $0 \le y \le 0.1$ and $-8 \le z \le 8$. Figures 4 and 5 show respectively, the nearfield and the farfield grid at the symmetry plane.

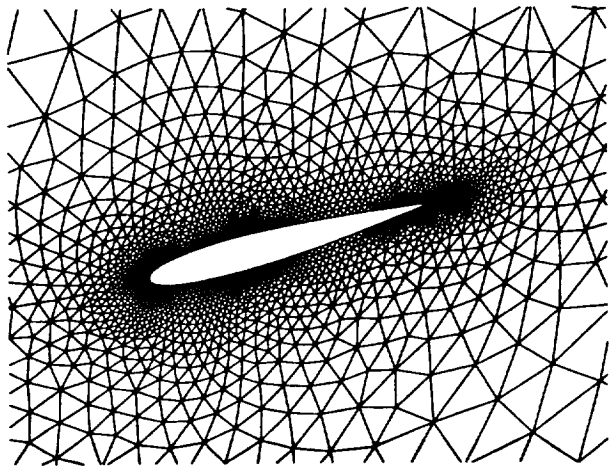

Figure 4: Near-field grid

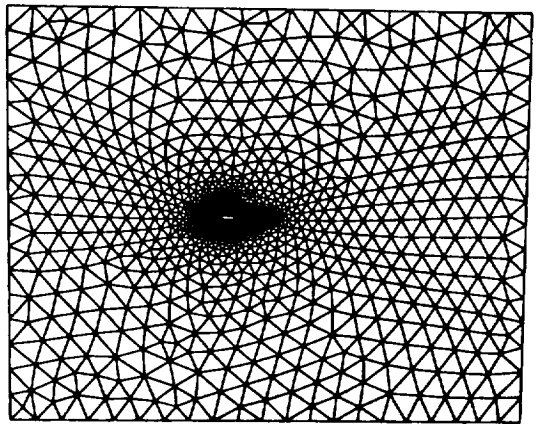

Figure 5: Far-field grid

Computations were carried out for the test cases of a) subsonic (subcritical) M=0.63, $\alpha = 2°$, b) transonic M=0.85, $\alpha = 1°$, and c) supersonic M=1.2, $\alpha = 0°$. Results are presented for the transonic flow case. Figure 6 shows the Iso-Mach number contours ($\Delta M = 0.05$) at middle of the wing (y=0.05). Figure 7 shows Cp distribution at same location. Both of these compare well with numerical work reported in reference [8]. It should be noted that the grid is relatively coarse as compared to the 2-D computational attempts. Results for the subsonic and the supersonic cases also provide similar accuracy. Performance of the solver over hetrogenous network topologies is under way and will be reported in the paper.

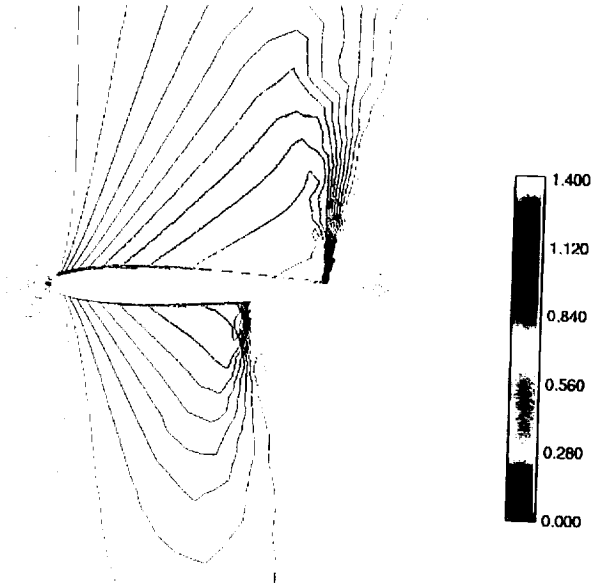

Figure 6: Iso-Mach contours

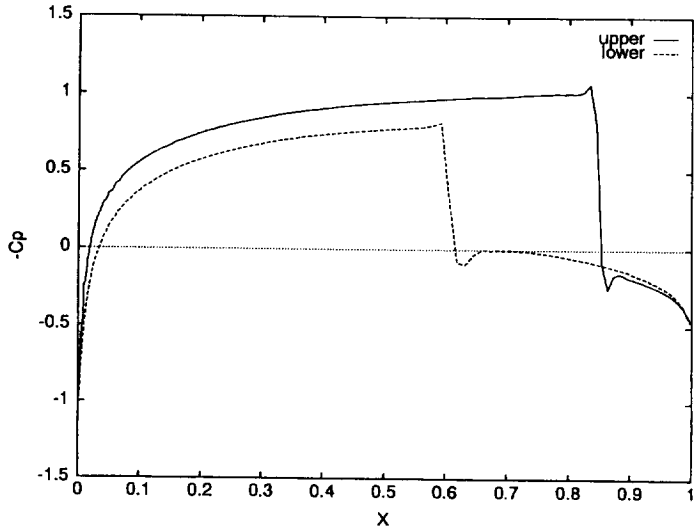

Figure 7: Cp distribution

5. References

1. Frink, N. T., "Recent Progress Towards a Three-Dimensional Unstructured Navier-Stokes Flow Solver," AIAA Paper 94-0061, 1994.

2. Frink, N. T., P. Parikh, and S. Pirzadeh, "A Fast Upwind Solver for the Euler Equations on Three-Dimensional Unstructured Meshes," AIAA Paper 91-0102, 1991.

3. Frink, N. T., "Upwind Scheme for Solving the Euler Equations on Unstructured Tetrahedral Meshes," AIAA Journal, Vol 30, No. 1, January 1992, pp 70-77.

4. Kwon, O. J., and C. Hah, "Simulation of Three-Dimensional Turbulent Flows on Unstructured Meshes," AIAA Journal, Vol 33, No. 6, June 1995, pp 1081-1089.

5. Singh, K. P., J. C. Newman, and O. Baysal, "Dynamic Unstructured Method for Flows past Multiple Objects in Relative Motion," AIAA Journal, Vol 33, No. 4, April 1995, pp 641-649.

6. Taylor, Stephen, Jerrell R. Watts, Marc A. Rieffel, and Michael Palmer, "The Concurrent Graph: Basic Technology for Irregular Problems," IEEE Parallel & Distributed Technology, Systems & Applications, pp. 15-25, Summer 1996.

7. Watts, Jerrell, and Stephen Taylor, "A Practical Approach to Dynamic Load Balancing," Technical Report, Scalable Concurrent Programming Laboratory, Syracuse University.

8. Viviand, H., "Numerical Solutions of 2D Reference Test cases," AGARD Technical Report AR-211, May 1985.

9. Delanaye, Michel, "Polynomial Reconstruction Finite Volume Schemes for the compressible Euler and Navier-Stokes Equations on Unstructured Adaptive Grids," Ph.D. Thesis, September 1996,University of Liege.

10. Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," Journal of Computational Physics, Vol 43, 1981.